# Path following and obstacle avoidance for an autonomous UAV using a depth camera

**Article** *in* Robotics and Autonomous Systems · May 2018

DOI: 10.1016/j.robot.2018.04.005

CITATIONS
19

READS
870

**2 authors:**

Massimiliano Iacono
Istituto Italiano di Tecnologia
**4** PUBLICATIONS **37** CITATIONS
SEE PROFILE

Antonio Sgorbissa
Università degli Studi di Genova
**174** PUBLICATIONS **1,493** CITATIONS
SEE PROFILE

# Path following and obstacle avoidance for an autonomous UAV using a depth camera

Massimiliano Iacono, Antonio Sgorbissa

**Abstract**

The main focus of this work is the development of a software architecture to autonomously navigate a flying vehicle in an indoor environment in presence of obstacles. The hardware platform used to test the developed algorithms is the AscTec Firefly equipped with a RGB-D camera (Microsoft Kinect): the sensor output is used to incrementally build a map of the environment and generate a collision-free path. Specifically, we introduce a novel approach to analytically compute the path in an efficient and effective manner. An initial path, given by the intersection of two 3D surfaces, is shaped around the obstacles by adding to either of the two surfaces a radial function at every obstacle location. The intersection between the deformed surfaces is guaranteed not to intersect obstacles, hence it is a safe path for the robot to follow. The entire computation runs on-board and the path is computed in real-time. In this article we present the developed algorithms, the software architecture as well as the results of our experiments, showing that the method can adapt in real time the robot's path in order to avoid several types of obstacles, while producing a map of the surroundings.

*Keywords:* UAV, MAV, Flying Vehicle, Obstacle avoidance, Path planning

## 1. Introduction

Multi-rotor copters have been widely adopted by the research community in the last few years because, in relation to other classes of aerial vehicles, they can be more easily controlled and have a high maneuverability. This allows them to navigate in small, human-unaccessible and inhospitable environments which makes them more suitable for indoor applications such as search and rescue operations or industrial inspection. Indoor navigation poses several problems which are not trivial to solve, like narrow maneuvering space and lack of GPS signal. In this work we aim at accomplishing obstacle avoidance and path following in an indoor environment using a flying robot equipped with a depth sensor. The vehicle has to perceive obstacles, map the environment, plan a collision-free path and then fly along it. Computational efficiency is a key issue for the project, since we need the entire system to run on-board. In this article, our main concern is the development and validation of the path planning and following algorithm, hence the robot localization is provided by a motion capture system. However, more realistic localization sources can be used without changing the other modules within the software architecture.

The main contributions of this work are:

- The design of an architecture for perception, mapping and navigation in an unknown indoor environment, based on the availability of a depth sensor returning obstacle information under the form of a point cloud.
- Implementation of a novel path planning approach, in which every element of the perceived point cloud is considered as an individual obstacle and contributes to the path deformation. The so computed path is guaranteed to be collision-free.
- Validation of the developed algorithms through several experiments.

The article is structured as follows: In Section 2 a Literature review is provided, in relation to the autonomous UAV navigation, the path planning algorithms and the control strategies; in Section 3 the algorithm for path planning is described, explaining how the path is defined and then deformed according to the perceived obstacles; in Section 4 we present the developed software architecture giving information about all the modules and the messages they exchange; in Section 5 the experiments are reported, providing details about the testing scenario and experimental outcomes; in Section 6 we discuss the results and provide conclusions.

## 2. Related works

### *2.1. Autonomous Navigation for UAVs*

In the past decade a lot of research has been performef on autonomous flying vehicles. Indeed there is a number of interesting applications in which an autonomous UAV can prove to be useful. Just to name some of them: search and rescue operations, area exploration, industrial inspection, surveillance and security, coverage of sports events and filming aerial shots. To accomplish such tasks, autonomous navigation is fundamental and a lot of different techniques to tackle this problem have been already investigated.

In [1] an AscTec Pelican[1] has been used to monitor the interior of a building subject to an earthquake with the support of

[1] `http://www.asctec.de/`

other mobile ground robots. Autonomous navigation has also been implemented on other AscTec platforms, such as the Firefly [2, 3], or the Hummingbird [4]. In [2], for instance a swarm of Firefly can autonomously perform optimal area coverage and mapping. In all of the afore mentioned works AscTec products were used. Such vehicles indeed provide a good trade-off between flying capabilities, computational power and weight, allowing for the development of real autonomous vehicles which can fly without the support of any external aid. Cheaper yet valid vehicles are produced by the Parrot company[2], such as the AR Drone, equipped with two cameras, pointing forward and downward. However the AR Drone does not mount an on-board computer hence it cannot navigate without the aid of a ground station. Nonetheless in [5] the AR Drone equipped with an external processing unit has been used to detect and follow a person, proving that autonomous navigation can be achieved even under very strict computational constraints.

In this work we have chosen to use the AscTec Firefly equipped with the MasterMind computer board, because of its good flying capabilities and computational resources. Additionally, it is easy to equip it with new sensors thanks to the numerous interfaces of the on-board computer.

As far as sensing capabilities are concerned, there are several valid choices regarding the sensor to mount on-board that have been investigated: single camera [6, 7, 8, 9], stereo camera [10, 11, 12, 13], depth camera [14, 15, 16] or Laser scanner [13, 17]. The chosen sensor must provide enough informations about the surroundings in order to correctly navigate the robot in presence of obstacles. In this work we used a depth camera, in particular a Microsoft Kinect, since it provides rich information with low power consumption, weight payload and computational resources.

### *2.2. Path Planning and Path Following for UAVs*

There is a number of possibilities to plan a path in presence of obstacles that have already been tested on wheeled robots and that can be extended to UAVs. Since the path needs to be updated in real-time one of the main requirements that the path planning algorithm must fulfill is computational efficiency, which is challenging due to the typically restricted processing capabilities of an on-board computer.

In [18] the use of Rapidly-exploring Random Trees is proposed, which can rapidly guarantee a collision-free path toward a goal. However, the path generated by the RRT is typically discontinuous and needs to be successively smoothed in order to prevent unnecessary maneuvering which are undesirable when working with flying robots.

The vehicle has to follow the generated path in the most responsive way and with as least oscillation as possible. A number of different approaches are presented in the Literature. The most used technique to cope with such problem is the classical PID controller. Indeed, most developers aiming at implementing an autonomous vehicle, either flying or not, make extensive use of such control algorithm, due to its implementation ease, versatility and low computational requirement. As far as flying robots are concerned, in [19] a PD controller has been implemented to keep the robot flying in the middle of a corridor, whereas the authors of [20] used a PID controller to autonomously navigate a Parrot AR Drone in a GPS-denied environment, along a previously computed collision-free path. Nevertheless there are other valid alternatives to PID control. In [16] a Model Predictive Control implementation is utilized to drive a MAV through a window. Model predictive control has the additional feature of taking into account a prediction of the future state in order to produce its output.

Alternatively, when using an approach that relies of vector fields to generate the path to be followed [13] there is no need for an additional control strategy, since the path planner automatically outputs the commands to drive the robot at every location in the space.

The most famous example of such algorithm is the Artificial Potential Field [21], which is fast and takes into account the presence of an arbitrary number of obstacles. Despite this, one well known issue of this approach is that it can get stuck in local minima. An example of Artificial Potential Field applied to UAV navigation is presented in [13]. In this work, to overcome the local minima problem, the authors had to use a multi-layered path planning approach, in which a global path is computed by the top layer using the A* algorithm [22], whereas the bottom layer locally implements obstacle avoidance using the Artificial Potential Field.

A novel approach that can simultaneously trace a collision-free path and generate control commands to drive the robot, while overcoming the local minima problem of APF (see Section 3 for further details), has been presented in [23]. The idea behind this algorithm is that a predefined path is deformed so that it does not intersect with obstacles. The initial path, analytically defined as the intersection of two 3D surfaces, is then shaped by adding to either of the two surfaces the contribution of surrounding obstacles, defined as a radial function centered in the obstacle (e.g., a Gaussian). The resulting surfaces intersection is guaranteed not to intersect the obstacles (see [23] for a detailed proof), hence it is a safe path for the robot to follow. The algorithm easily scales up, so it can be used with an arbitrary number of obstacles (e.g. a point cloud in which every point is considered as a small obstacle). The vehicle is then driven toward and along the reshaped path by defining a velocity vector which can be easily computed at any given pose of the robot. However, when controlling a multicopter, an additional policy is required to control the yaw of the vehicle. Generally speaking, the desired robot's orientation should face the direction towards which it is moving, especially in the case of forward looking sensors (this is not true if we wish the vehicle to track some feature of interest which is not in front of it). To achieve this, it is reasonable to set the target yaw as corresponding to the current direction of motion, and using one of the afore mentioned control methods (e.g., a PD controller) to rotate the vehicle.

In previous works [24, 25] the algorithm has been successfully applied to a flying robot to avoid obstacles of known size. The obstacles were perceived detecting the visual ArUco mark-

[2] `https://www.parrot.com`

ers applied onto them (synthetic square markers composed by a wide black border and a inner binary matrix which determines its identifier) and the algorithm required to know the obstacles size beforehand. Such works have proven the algorithm applicability, but using markers is obviously unfeasible in real-world environments. In this work we want to implement the proposed method in a realistic scenario, with the robot navigating in an unknown environment. To this end we have equipped the vehicle with a Microsoft Kinect sensor used to detect any kind of unlabeled obstacle. Each element of the perceived point cloud is then treated as an individual obstacle, also showing the scalability of the algorithm to an arbitrary number of obstacles. The resulting system allows for simultaneous environment mapping and obstacle avoidance, in a real-time fashion. This required to solve a number of problems related to real-time mapping and obstacle avoidance by using on board computational resources, that were ignored in [24, 25] and are tackled here for the first time.

## 3. Path Planning Algorithm

The algorithm presented in [23] allows for path planning in presence of an arbitrary number of obstacles by deforming a predefined path described as the intersection of two surfaces.

### *3.1. Path definition and following*

The desired path is defined as the intersection of two 3-dimensional surfaces as follows:

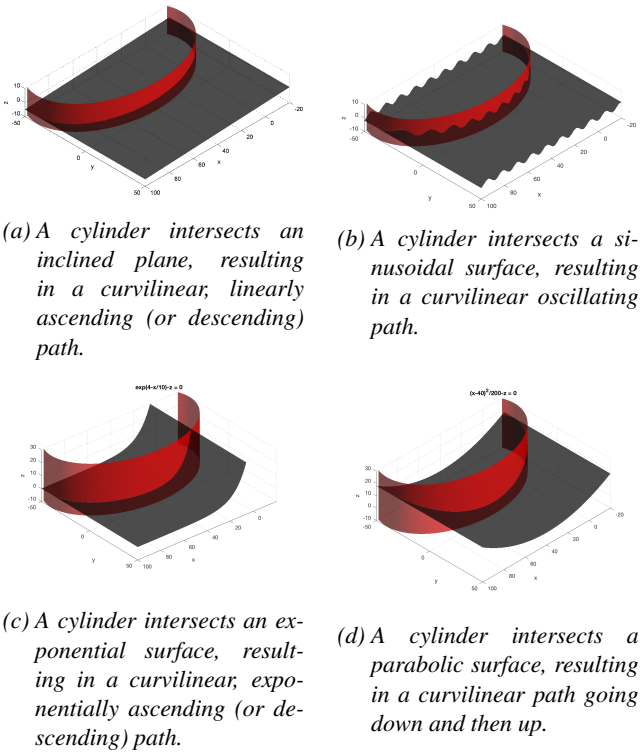
$$\begin{cases} f_1(x,y,z) = 0 \\ f_2(x,y,z) = 0 \end{cases} \tag{1}$$

Figure 1a shows an example of a path defined in this way. The two functions in equation (1) must fulfill the following requirements:

- They must be twice differentiable
- Their gradient norm must be non-null in $\mathbf{R}^3 \setminus \mathbf{D_i}$, where $\mathbf{D_i}$ is a set of points for which $\|\nabla f_i\|^2 = 0, i = 1, 2$
- Their gradient cross product must be non-null, that is the two gradient vectors must not be parallel in $\mathbf{R}^3 \setminus \mathbf{D_{12}}$ where $\mathbf{D_{12}}$ is a set of points not belonging to the path, for which $\nabla f_1 \times \nabla f_2 = 0$
- the gradient $\nabla f_1(x,y,z)$ is never parallel to the z-axis, that is $f_1(x,y,z)$ is mostly used to drive the vehicle along the $xy$ plane
- The gradient $\nabla f_2(x,y,z)$ is never parallel to the $xy$ plane, that is $f_2(x,y,z)$ is mostly used to keep the vehicle at the right altitude

The velocity vector $v(x,y,z)$ which drives the vehicle toward the desired path from any position $(x,y,z)$ is defined as:

$$v(x,y,z) = -K_{grad_1} f_1 \nabla f_1 - K_{grad_2} f_2 \nabla f_2 + K_{tang}(\nabla f_1 \times \nabla f_2) \tag{2}$$



*(a) A cylinder intersects an inclined plane, resulting in a curvilinear, linearly ascending (or descending) path.*

*(b) A cylinder intersects a sinusoidal surface, resulting in a curvilinear oscillating path.*

*(c) A cylinder intersects an exponential surface, resulting in a curvilinear, exponentially ascending (or descending) path.*

*(d) A cylinder intersects a parabolic surface, resulting in a curvilinear path going down and then up.*

*Figure 1: Example of surfaces intersections resulting in different 3D paths.*

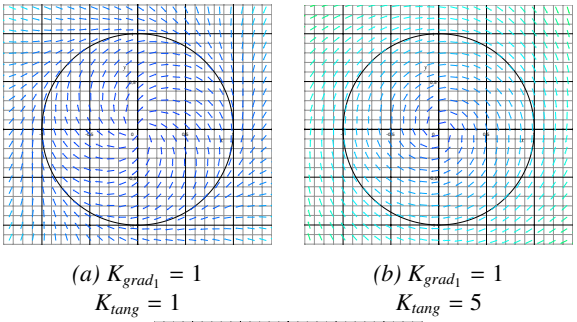
With an abuse of notation in equation (2) we refer to $f_i(x,y,z)$ as $f_i$ and to $\nabla f_i(x,y,z)$ as $\nabla f_i$ for the sake of conciseness.

At this point we can provide an intuitive explanation to (2) by giving an interpretation to each term:

- Both functions $f_i(x,y,z)$, $i = 1, 2$ describe a surface that divides the space in two subspaces such that the functions are null if and only if $(x,y,z)$ belongs to the surface. In any other case the functions value represents the signed distance from the surface, negative when $(x,y,z)$ belongs to one subspace and positive when it belongs to the other. Which subspace is positive or negative depends on the function choice.
- The negative gradients $-\nabla f_1(x,y,z)$ and $-\nabla f_2(x,y,z)$ define two vectors perpendicular to the level surfaces of $f_1(x,y,z)$ and $f_2(x,y,z)$ respectively, passing through $(x,y,z)$, i.e., they define the direction along which the vehicle should move to get closer to the path. Multiplying these vectors with the function value at any $(x,y,z)$ results in a vector directed toward the path whose length is proportional to the distance from the surface.
- $\nabla f_1(x,y,z) \times \nabla f_2(x,y,z)$ denotes a vector perpendicular to the two gradients. This results in a vector tangent to both level surfaces at any $(x,y,z)$.
- The final velocity vector is defined in (2) as a combination of the gradient and tangent contributions producing a vector field whose vectors converge toward and along the

desired path. In fact when the vehicle moves exactly on the desired path $f_1(x, y, z) = f_2(x, y, z) = 0$, hence the gradient contribution is null and the robot is driven tangentially to the surfaces intersection.

- $K_{grad_i}$ and $K_{tang}$ are tunable parameters that determine the contribution of gradient and tangent vectors respectively. Figure 2 provides a visualization of the effects of these parameters. We can see how $K_{grad_i}$ increases the convergence speed but, if too high, may cause oscillations due to abrupt direction changes in the proximity of the path. Instead, $K_{tang}$ helps to make the vehicle approach the path more smoothly, but it can also increase the time required to converge.



(a) $K_{grad_1} = 1$, $K_{tang} = 1$ (b) $K_{grad_1} = 1$, $K_{tang} = 5$ (c) $K_{grad_1} = 5$, $K_{tang} = 1$

*Figure 2: Top view of vector fields generated by equation* (2) *with* $f_1(x, y, z) = x^2 - y^2 - 1$ *and* $f_2(x, y, z) = z - 1$*, resulting in a circular path at constant height. Vector fields with different values of* $K_{grad_1}$ *and* $K_{tang}$ *are shown. The lighter its color, the longer the vector. For visualization purposes the vector field is projected on the XY plane, hence the effect of* $K_{grad_2}$ *is not visible. We can see that higher values of* $K_{grad_1}$ *(2c) cause the vectors to change abruptly direction in the proximity of the path eventually resulting in oscillations around the path in experiments, whereas higher values of* $K_{tang}$ *(2b) significantly reduces the convergence speed towards the desired path. A proper tuning of these parameters is fundamental for a smooth and fast path following (2a).*

**Remark 1.** *Unlike APF, the Equation* (2) *used to compute the velocity command to drive the robot is not the result of attractive and repulsive contribution exerted by goal and obstacles respectively. Additionally it cannot lead to deadlocks, even in presence of obstacles, overcoming the local minima problem of APF. A rigorous proof of this claim is provided in [23].*

### *3.2. Path deformation in presence of obstacles*

The desired path must be modified in presence of obstacles in order to avoid them. To do so either of the two surfaces introduced in (1) must be deformed so that their intersection does not pass through the obstacles. Without loss of generality we can arbitrarily choose to deform either $f_1(x, y, z)$ or $f_2(x, y, z)$. Clearly, this choice affects the resulting path so that deforming $f_1(x, y, z)$ will drive the robot aside the obstacles, whereas operating on $f_2(x, y, z)$ will make it fly above or below them. From now on we consider $f_1(x, y, z)$ as the function to be modified.

Each obstacle $j$ is defined by its position in space $(x_j, y_j, z_j)$ and size $r_j$. In particular, an individual obstacle is defined as a spherical neighborhood:

$$O_j = \{(x, y, z) \; s.t. \; |(x, y, z) - (x_j, y_j, z_j)| < r_j\} \tag{3}$$

The deformation is performed by adding to $f_1(x, y, z)$ an obstacle function $O_j(x, y, z)$ for every obstacle $j$ yielding a new surface:

$$f_1'(x, y, z) = f_1(x, y, z) + \sum_j O_j(x, y, z) \tag{4}$$

that, properly intersected with $f_2(x, y, z) = 0$ will produce the deformed path (1).

$O_j(x, y, z)$ has to fulfill the following requirements:

- $f_1'(x, y, z)$ is guaranteed not to pass through the obstacle.
- Let $d_j$ be the euclidean distance between the vehicle and the obstacle $j$, and $d_{max} > 0$ a value which defines the obstacles range of influence. Then $O_j(x, y, z)$ must fulfill the condition:

$$\begin{cases} O_j(x, y, z) \neq 0 & d_j < d_{max} \\ O_j(x, y, z) = 0 & d_j \geq d_{max} \end{cases} \tag{5}$$

For instance, a possible choice of $O_j(x, y, z)$ can be a bell-shaped function with its maximum in the center of the obstacle and monotonically decreasing to 0 when getting farther from it. Our choice of $O_j$ is:

$$\begin{cases} O_j(x, y, z) = A_j(1 + cos(\pi d_j(x, y, z)/\sigma)) & d_j < \sigma \\ O_j(x, y, z) = 0 & d_j \geq \sigma \end{cases} \tag{6}$$

where $\sigma$ is a tunable parameter which determines the distance at which the obstacles start to be perceived.

Instead, the amplitude of the obstacle $A_j$ must be computed for every obstacle $j$ in order to guarantee that the new function $f_1'(x, y, z)$ actually describes a collision-free path. For the latter assumption to hold it must be true that:

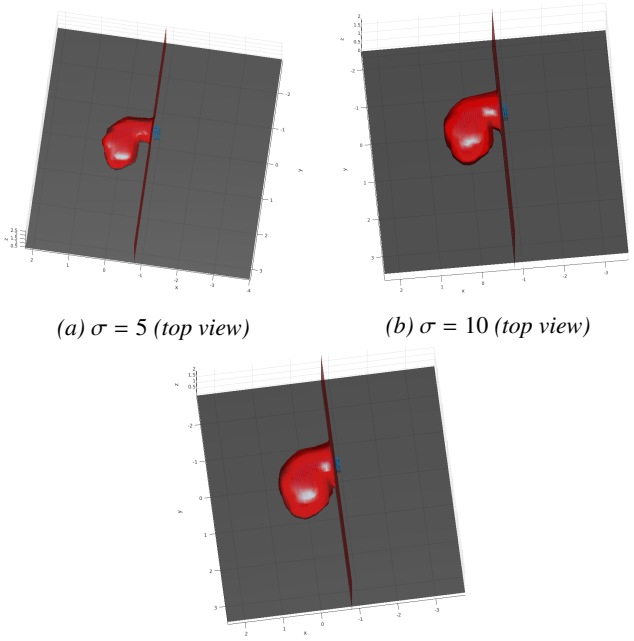
$$f_1'(x, y, z) \neq 0 \;\; \forall (x, y, z) \in O \tag{7}$$

where $O$ is the obstacle region, that is the area of space occupied by obstacles plus a safety margin around them adjusted with the $\sigma$ parameter. In (7) we are imposing that the two surfaces $f_2(x, y, z) = 0$ and $f_1'(x, y, z) = 0$ must not intersect inside the

obstacle region. To ensure this, we can tune the amplitude of the bell function $A_j$ according to the obstacle size $r_j$. In [23] it is shown how to compute $A_j$ to ensure that the deformed path dose not intersect any obstacle by staying as close as safety allows to the obstacles, and it is proven that the path maintains its continuity after the deformation, since it is defined as the sum of continuous surfaces.

Unlike approaches exploiting Artificial Potential Field which generate a vector field given the goal and the obstacles, with this technique we can define a path and then modify it to obtain a collision-free path and, due to its continuity, we prevent the risk of being stuck in local minima. Once again, a formal proof of this concept is provided in [23], however, this can be briefly explained as follows: if two continuous surfaces intersect (either they are closed or open infinite surfaces), their intersection is necessarily a curve with no endpoints. Any local modification of either surface, by adding a sum of continuous contributions, will preserve their continuity, and therefore their intersection will always be a continuous curve with no endpoints. As a consequence, any vector field that drives the robot towards and along such curve will have no local minima. Figure 1 shows some examples of 3D paths that can be defined as surfaces intersection. Using this method we can drive the robot towards any 3D path simply changing the two intersecting surfaces. Figure 4 shows how the presence of a single obstacle (Figure 4a) or multiple obstacles forming a maze (Figure 4b) deforms the red surface to generate a collision-free path even in a very complex environment.

In Figure 3 it is shown how the same path gets deformed by a point cloud belonging to an L-shaped obstacle with different values of $\sigma$. Each point in the cloud is considered as a small individual obstacle and the sum of their contribution generates the final path which traces a path all around the obstacle. Higher values of $\sigma$ make the path longer but safer in terms of distance from the obstacle, whereas a smaller $\sigma$ causes the robot to fly closer to the obstacles along a shorter path. Indeed $\sigma$ can be considered as a safety margin, which can be tuned depending on how conservative the resulting path should be.



*(a)* $\sigma = 5$ *(top view)* *(b)* $\sigma = 10$ *(top view)*

*(c)* $\sigma = 15$ *(top view)*

*Figure 3: In the Figure the two surfaces which define the path are shown. They were originally two planes, then the red surface has been deformed due to the presence of the obstacle, represented by a point cloud (small blue dots). The intersection between these two new surfaces does not intersect with the obstacle and traces a way to pass aside it. The deformation is performed with different values of $\sigma$ showing how this parameter influences the resulting path. Higher values of $\sigma$ generate a longer but safer path, whereas lower values result in shorter path with the robot flying closer to the obstacles.*

## 4. Software Architecture

In Figure 5 the overall software architecture is shown. The entire architecture is developed on the ROS middleware [26] in order to make it more portable and easier to expand in future works. In the Figure, the ROS nodes are represented as squared boxes, whereas the arrows are labeled with the input/output of each module.

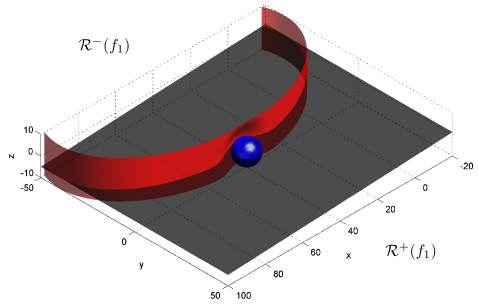
The point cloud is acquired by the Kinect, then subsampled and fed into the Octomap server [27], in charge of incrementally building the map of the robot vicinity. The choice of the Octomap as a data structure for the obstacle map is motivated by the efficiency with which the map can be patched with new data due to its tree structure which is fast to query. This is particularly convenient for our application in which the environment map is incrementally built from scratch. The Path Planner node then takes as input the map together with the robot pose, coming from the Motion capture (MoCap) system, to generate the velocity commands as explained in Section 3. The commands are then fed to the robot via the AscTec interface which controls the multicopter motors.

## 5. Experiments

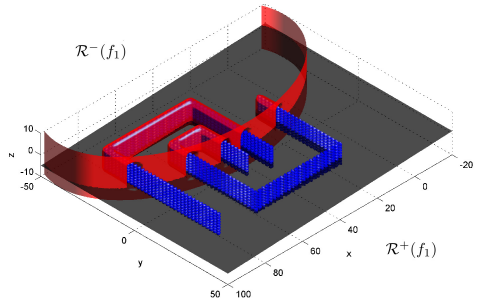
We have experimented the presented algorithm on the AscTec Firefly vehicle, equipped with the Microsoft Kinect sensor. The robot mounts the Mastermind on-board, which is a light and powerful computational board with an embedded Intel i7 processor, 4GB of RAM, 64GB of SSD storage memory and the most classical interfaces such as USB and Ethernet ports. The system has been tested under various conditions to prove the robustness of the proposed method. The results are then visualized in Figures 6, 7, 8 showing both the generated Octomap and the followed path [3].

In the experiments the robot was asked to follow a straight path at fixed height (intersection between an horizontal and a vertical plane) along which different types of obstacles were

[3] Some videos showing the experimental results can be found here: https://drive.google.com/open?id=0BzSRFMNKnL5GNkpLTjRfUzM3bmc




*(a) Ellipsoidal path deformed in presence of a single obstacle*




*(b) Ellipsoidal path deformed in maze-like environment.*

*Figure 4: Ellipsoidal path deformed in different scenarios. Even in a very complex environment (4b) the algorithm is still capable of finding a collision-free path around the obstacles.*

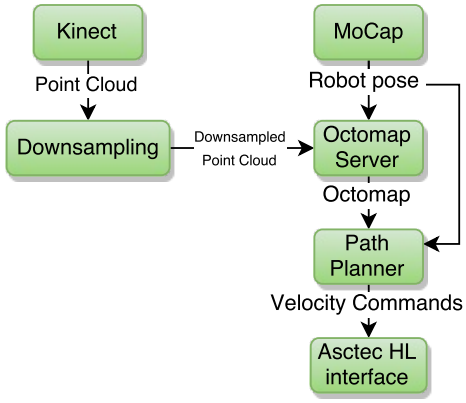
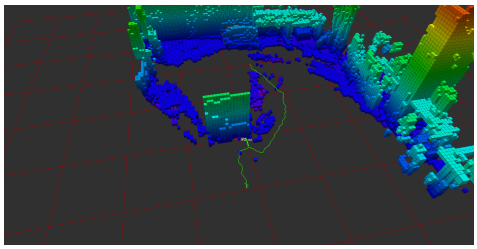
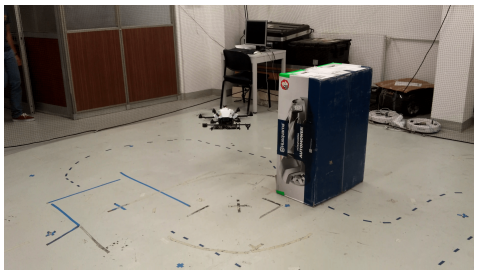



*Figure 5: Overall Software Architecture. The boxes contain the software modules, the arrows are labeled with the output of each module*

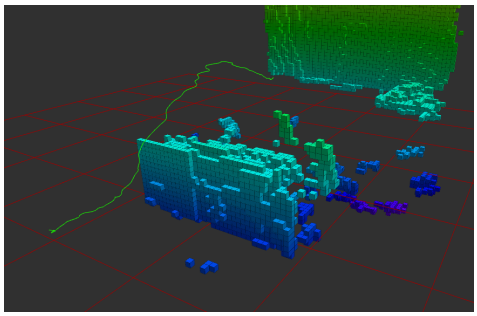


*(a) Octomap*

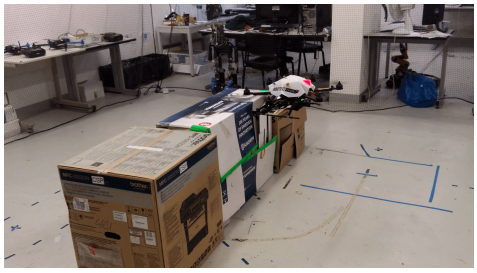


*(b) The robot flying around the obstacle*

*Figure 6: Experiment with a box-shaped obstacle*



*(a) Octomap*

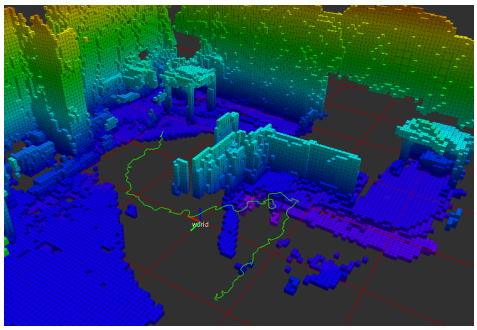


*(b) The robot flying above the obstacle*

*Figure 7: Experiment with a barrier obstacle*

placed. In order to avoid sudden and scattering maneuvers, the path planner output is low-pass filtered by averaging the commands sent in the last 3 seconds (one can tune the length of this temporal window to increase or decrease the robot responsiveness). It is worthwhile mentioning that the robot has no initial knowledge about the environment. It can be observed in the Figures that not all of the surrounding is mapped, since during the experiment only a part of the environment comes in the visual range of the sensor. For better perception, the yaw is controlled so that the robot, hence the Kinect sensor, always faces the direction in which it is moving. To achieve this a classical PD controller is implemented on the yaw command, providing as desired yaw angle $\theta_d$ the direction of the velocity vector $v$ which drives the robot (2). This prevents the robot from flying in areas that have not been mapped before. For these experiments $K_{grad_1}$ and $K_{grad_2}$ have been set to 0.1 and $K_{tang}$ to 0.7. These values have been empirically chosen to ensure a good trade off between convergence time and smoothness. We present three experiments:

- In a first experiment a single box has been placed along the trajectory. (Figure 6). The robot started in front of the



*(a) Octomap*

*(b) The robot flying around the obstacle*

*Figure 8: Experiment with a L-shaped obstacle*

box, arrived in the obstacle range of influence and steered to its right in order to avoid it.

- In a second experiment several boxes have been placed in order to form a barrier (Figure 7). In this case we wanted to show how the robot behaves when $f_2(x, y, z)$ is deformed rather than $f_1(x, y, z)$ (refer to Section 3). In this case the vehicle increases its thrust to fly above the obstacle and then flies down beyond it.

- To make things more difficult we wanted to test a L-shaped obstacle (Figure 8), which has always been an issue for approaches such as Artificial Potential Field, since in the middle of the "L" it is very likely to end up in a local minimum. Once again, the obstacle is successfully avoided tracing a path all the way around it.

When a huge number of data come into play it is of fundamental importance to keep sending commands at a high rate. The minimum frequency at which velocity commands can be generated is 10Hz, otherwise they will be ignored[4]. In these experiments we have measured a publishing rate of the commands varying from 20 to 100 Hz, depending on the size of the map processed at that particular moment, whereas the map is updated at a frequency ranging between 1 and 8 Hz. Please remark that the achievable publishing rate is strictly dependent on the computational requirements of the algorithm. Indeed, a publishing rate of 20Hz means that the algorithm manages to perform all the computations required in (4) (summing over all occupied voxels to deform the original surface) and (2) (compute the velocity command) within 50ms.

Figure 9 reports four graphs showing the evolution over time of the command publishing rate with respect to the map size. Each graph shows the result of a different experiment. It can be observed that the performance decreases as the map gets bigger, nevertheless the commands are generated fast enough to drive the robot (minimum rate 20Hz). It has to be considered that these experiments have been carried out with a rather high Octomap resolution (voxel size of 5cm). In order to reduce the computational requirements one can either decrease the Octomap resolution or opt for different approaches, such as clearing areas of the map which are far away from the robot, or likely not to be explored anymore. We have not taken into account these problems in our study, since we have been working in a narrow workspace, but these issues should be taken into account in order to implement the approach on a greater scale.

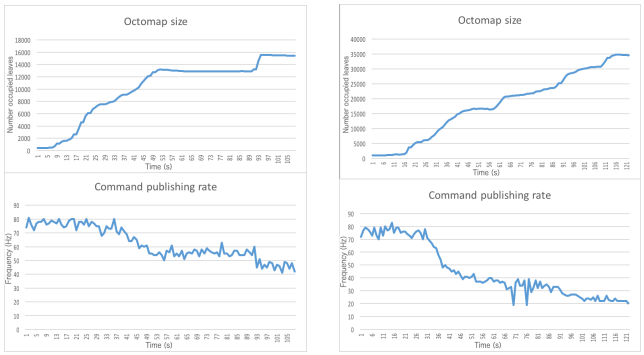



*Figure 9: Publishing command rate over time with respect to the size of the map. Every graph shows the result of a different experiment.*

## 6. Discussion and Conclusions

The experiments presented in Section 5 show that the developed algorithm is a valid technique to navigate a flying robot in presence of obstacles in a previously unknown environment.

Specifically, our method presents some advantages that are worthwhile mentioning:

- The path is defined analytically, hence it is easy to compute the gradient and the tangent vectors, used to control the robot (2).

- It is computationally efficient, hence it can run even on a computer installed on-board of small flying vehicles.

- It is scalable up to an arbitrary number of obstacles, even though when increasing the map size it should be considered that the efficiency may decrease.

- It can adapt to change in the environment. In fact, since the path is deformed in real-time the robot will avoid new obstacles and tend to recover the original path when an already perceived obstacle is removed.

[4]Check the documentation of the `asctec_hl_interface` at `http://wiki.ros.org/asctec_hl_interface`

The only limitation that prevents the system from being fully autonomous, is that it relies on the Motion Capture system for localization. We are aiming at getting rid of the dependency on the MoCap by implementing one of the already available localization algorithms, such as RGB-D SLAM [28].

At the moment the only obstacles being considered by our system are the one which are labeled as occupied by the Octomap server. As a future work, one may take into account the unknown space as well, in order to deal with the problem of "obstacles around the corner". In fact, as explained in Section 5, only the obstacles within sensor range are added to the map. Another possibility to overcome this problem is to embed additional sensors to scan a wider area and merge their data together.

Finally, in our implementation the surface to be deformed is decided in advance, as well as the steering direction when avoiding an obstacle (left/right, up/down). A method to compute at run-time how to set these parameters, in order to dynamically adapt to different obstacle configurations, shall be explored.

## Acknowledgments

The work has been supported by MIUR (Italian Ministry of Education, Universities and Research) through the project DIONISO (Seismic domotics innovative technologies for home and system safety, Smart cities and communities and Social Innovation).

## References

[1] N. Michael, S. Shen, K. Mohta, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, S. Tadokoro, Collaborative mapping of an earthquake damaged building via ground and aerial robots, Springer Tracts in Advanced Robotics 92 (2014) 33–47, doi:10.1007/978-3-642-40686-7_3.

[2] D. Scaramuzza, M. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, L. Meier, Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in GPS-denied environments, IEEE Robotics and Automation Magazine 21 (3) (2014) 26–40, doi:10.1109/MRA.2014.2322295.

[3] S. Weiss, M. Achtelik, M. Chli, R. Siegwart, Versatile distributed pose estimation and sensor self-calibration for an autonomous MAV, 31–38, doi:10.1109/ICRA.2012.6225002, 2012.

[4] M. Blsch, S. Weiss, D. Scaramuzza, R. Siegwart, Vision based MAV navigation in unknown and unstructured environments, 21–28, doi:10.1109/ROBOT.2010.5509920, 2010.

[5] J. Lugo, A. Zell, Framework for autonomous on-board navigation with the AR.Drone, Journal of Intelligent and Robotic Systems: Theory and Applications 73 (1-4) (2014) 401–412, doi:10.1007/s10846-013-9969-5.

[6] J.-O. Lee, K.-H. Lee, S.-H. Park, S.-G. Im, J. Park, Obstacle avoidance for small UAVs using monocular vision, Aircraft Engineering and Aerospace Technology 83 (6) (2011) 397–406, doi:10.1108/00022661111173270.

[7] X. He, Z. Cai, D. Huang, Y. Wang, Indoor navigation for aerial vehicle using monocular visual SLAM, 2071–2075, doi:10.1109/CGNCC.2014.7007496, 2015.

[8] G. Chowdhary, E. Johnson, D. Magree, A. Wu, A. Shein, GPS-denied indoor and outdoor monocular vision aided navigation and control of unmanned aircraft, Journal of Field Robotics 30 (3) (2013) 415–438, doi:10.1002/rob.21454.

[9] S. Weiss, D. Scaramuzza, R. Siegwart, Monocular-SLAM-based navigation for autonomous micro helicopters in GPS-denied environments, Journal of Field Robotics 28 (6) (2011) 854–874, doi:10.1002/rob.20412.

[10] K. Schauwecker, A. Zell, On-board dual-stereo-vision for the navigation of an autonomous MAV, Journal of Intelligent and Robotic Systems: Theory and Applications 74 (1-2) (2014) 1–16, doi:10.1007/s10846-013-9907-6.

[11] L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, M. Pollefeys, PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision, Autonomous Robots 33 (1-2) (2012) 21–39, doi:10.1007/s10514-012-9281-4.

[12] L. Garca Carrillo, A. Dzul Lpez, R. Lozano, C. Pgard, Combining stereo vision and inertial navigation system for a quad-rotor UAV, Journal of Intelligent and Robotic Systems: Theory and Applications 65 (1-4) (2012) 373–387, doi:10.1007/s10846-011-9571-7.

[13] M. Nieuwenhuisen, D. Droeschel, M. Beul, S. Behnke, Obstacle detection and navigation planning for autonomous micro aerial vehicles, 1040–1047, doi:10.1109/ICUAS.2014.6842355, 2014.

[14] R. Leishman, T. McLain, R. Beard, Relative navigation approach for vision-based aerial GPS-denied navigation, Journal of Intelligent and Robotic Systems: Theory and Applications 74 (1-2) (2014) 97–111, doi:10.1007/s10846-013-9914-7.

[15] Q. Li, D.-C. Li, Q.-F. Wu, L.-W. Tang, Y. Huo, Y.-X. Zhang, N. Cheng, Autonomous navigation and environment modeling for MAVs in 3-D enclosed industrial environments, Computers in Industry 64 (9) (2013) 1161–1177, doi:10.1016/j.compind.2013.06.010.

[16] G. Flores, S. Zhou, R. Lozano, P. Castillo, A vision and GPS-based real-time trajectory planning for a MAV in unknown and low-sunlight environments, Journal of Intelligent and Robotic Systems: Theory and Applications 74 (1-2) (2014) 59–67, doi:10.1007/s10846-013-9975-7.

[17] S. Shen, N. Michael, V. Kumar, Autonomous multi-floor indoor navigation with a computationally constrained MAV, 20–25, doi:10.1109/ICRA.2011.5980357, 2011.

[18] H. Yu, R. Beard, A vision-based collision avoidance technique for micro air vehicles using local-level frame mapping and path planning, Autonomous Robots 34 (1-2) (2013) 93–109, doi:10.1007/s10514-012-9314-z.

[19] S. Zingg, D. Scaramuzza, S. Weiss, R. Siegwart, MAV navigation through indoor corridors using optical flow, 3361–3368, doi:10.1109/ROBOT.2010.5509777, 2010.

[20] I. Sa, H. He, V. Huynh, P. Corke, Monocular vision based autonomous navigation for a cost-effective MAV in GPS-denied environments, 1355–1360, doi:10.1109/AIM.2013.6584283, 2013.

[21] S. Ge, Y. Cui, Dynamic motion planning for mobile robots using potential field method, Autonomous Robots 13 (3) (2002) 207–222, doi:10.1023/A:1020564024509.

[22] P. Hart, N. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107, doi:10.1109/TSSC.1968.300136.

[23] A. Sgorbissa, Integrated Robot Planning, Obstacle Avoidance, and Path Following in 2D and 3D: ground, aerial, and underwater vehicles doi:10.13140/RG.2.2.14838.80969.

[24] P. Nguyen, C. Recchiuto, A. Sgorbissa, Real-time path generation for multicopters in environments with obstacles, vol. 2016-November, 1582–1588, doi:10.1109/IROS.2016.7759256, 2016.

[25] P. Nguyen, C. Recchiuto, A. Sgorbissa, Real-Time Path Generation and Obstacle Avoidance for Multirotors: A Novel Approach, Journal of Intelligent and Robotic Systems: Theory and Applications 89 (1-2) (2018) 27–49, doi:10.1007/s10846-017-0478-9.

[26] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: an open-source Robot Operating System, in: ICRA Workshop on Open Source Software, 2009.

[27] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, OctoMap: An efficient probabilistic 3D mapping framework based on octrees, Autonomous Robots 34 (3) (2013) 189–206, doi:10.1007/s10514-012-9321-0.

[28] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard, An evaluation of the RGB-D SLAM system, 1691–1696, doi:10.1109/ICRA.2012.6225199, 2012.